
Programación en C

DATSI, FI, UPM

José M. Peña

`jmpena@fi.upm.es`

Índice

- Estructura de un programa C.
- Variables básicas.
- Operaciones aritméticas.
- Sentencias de control.
- Arrays y Strings.
- Funciones.
- Estructuras de datos.
- Entrada/Salida básica.
- Ejemplos I.
- Modificadores de ámbito de las variables.
- Punteros y memoria dinámica.
- Operadores de bit.
- Preprocesador C y compilación.
- Librerías estándar.
- Ejemplos II.

Programación en C

Estructura de un programa en C

Estructura de un programa en C

Función `main()`:

```
int main( )  
{  
    printf( "Hola mundo!!\n" );  
    return( 0 );  
}
```

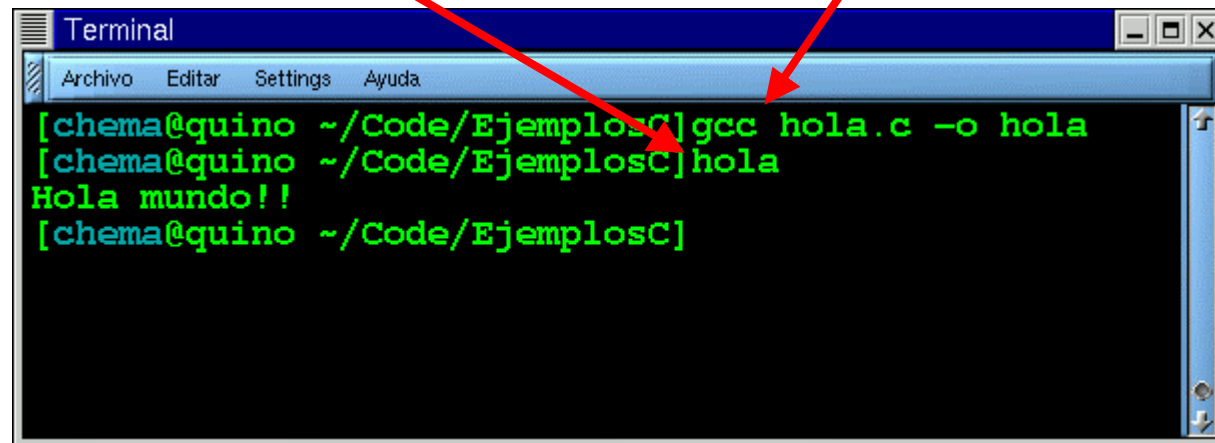
Estructura de un programa en C

Fichero hola.c

```
int main()  
{  
    printf("Hola mundo!!\n");  
    return(0);  
}
```

Compilación

Ejecución



A terminal window titled "Terminal" with a menu bar containing "Archivo", "Editar", "Settings", and "Ayuda". The terminal shows the following commands and output in green text on a black background:

```
[chema@quino ~/Code/EjemplosC] gcc hola.c -o hola  
[chema@quino ~/Code/EjemplosC] hola  
Hola mundo!!  
[chema@quino ~/Code/EjemplosC]
```

Two red arrows point to the terminal: one from the word "Ejecución" pointing to the command `hola`, and another from the word "Compilación" pointing to the command `gcc hola.c -o hola`.

Características de C

- Sensible a mayúsculas y minúsculas: `sum` y `Sum`
- Indentación y espacios en blanco.
- Sentencias (terminan con un punto y coma).
- Bloques (delimitados entre llaves).
- Elementos de un programa:
 - Palabras reservadas (muy pocas).
 - Funciones de librería estándar.
 - Variables y funciones definidas por el programador.

Finalización del programa

- Se llega al final de la función `main ()`.
- La función `main ()` realiza una llama `return ()`.
- Cualquier función realiza la llamada `exit ()`.

```
int main( )
{
    exit(0);
    printf( "Esto no se ejecuta\n" );
}
```

Comentarios

- Los comentarios en C pueden ocupar varias líneas y se encuentran delimitados entre `/*` y `*/`.

```
int main( )  
{  
    /* Esto es un comentario de varias  
        líneas.*/  
    return(0);  
}
```

Programación en C

Variables básicas

Tipos de variables

- Los tipos elementales de variables en C son:
 - Enteros (`int`).
 - Reales (`float`, `double`).
 - Caracteres (`char`).
 - Punteros (*).

NO existe un tipo booleano (en su lugar se usa `int` o `char`).

Modificadores de tipos

- Ciertos tipos básicos admiten diversos modificadores:
 - `unsigned` : Sólo valores positivos (sin signo).
 - `signed` : Valores positivos y negativos (por omisión).
 - `long` : Formato largo (para enteros únicamente).

Ejemplos:

`unsigned int`

`signed char`

`long int` (usualmente representado como `long`)

`unsigned long int`

Declaración de variables

- Declaración simple:
 - `char c;`
 - `unsigned int i;`
- Declaración múltiple:
 - `char c,d;`
 - `unsigned int i,j,k;`
- Declaración y asignación:
 - `char c='A';`
 - `unsigned int i=133,j=1229;`

Llamada sizeof ()

- La llamada sizeof() se utiliza para determinar el número de bytes que ocupa una variable o un tipo:

```
int a;
```

```
sizeof(a);
```

```
sizeof(unsigned int);
```

Ámbito de las variables

- La declaración de las variables lleva asociado un ámbito, dentro del cual la variable es visible:
 - Ámbito global: La variable es visible para todas las funciones del programa.
 - Ámbito local: La variable es visible sólo dentro de la función. (Tiene prioridad sobre el ámbito global)

Ámbito de las variables

```
int x,y;
int main( )
{
    float x,z;
    /* Aquí x y z son reales e y un entero */
}

/* Aquí x e y son variables enteras */
/* La variable z no existe fuera de la
función */
```

Expresiones constantes

- El formato de las expresiones constantes es;
 - Un expresión real se puede dar tanto en notación decimal (2.56) como científica (2.45E-4).
 - A una expresión de tipo long se le añade un L al final (200L).
 - Una expresión de tipo carácter se define entre comillas simples ('A').

Expresiones constantes

- Para definir las constantes de tipo carácter asociadas a caracteres especiales se usan secuencias de escape:
 - `'\n'`: Retorno de carro.
 - `'\t'`: Tabulador.
 - `'\b'`: Bell.
 - `'\0'`: Carácter nulo.
 - ...

Expresiones constantes

- Las constantes enteras se pueden representar en diferentes bases numéricas:
 - Base decimal: 230.
 - Base hexadecimal: 0x3A0 (comienza por cero-x).
 - Base octal: 0210 (comienza por cero).

Punteros

- Las variables de tipo puntero representan direcciones donde almacenar valores. Es importante diferenciar entre puntero (espacio de memoria donde se almacena la dirección) y la propia dirección apuntada por el puntero (su valor).
- Se declaran con un asterisco delante del identificador de la variable:

```
int *px,y;    /* px es un puntero a entero e y  
              un entero */
```

Punteros

- Los gestión de punteros admite dos operadores básicos:
 - Si `px` es un puntero (dirección): `*px` es el contenido del puntero (el valor almacenado en la dirección).
 - Si `x` es una variable: `&x` es la dirección de memoria donde está almacenada la variable.

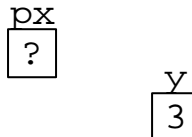
Punteros

```
int main()  
{  
    int *px,y=3;  
  
    px=&y;  
    /* px apunta a y */  
  
    *px=5;  
    /* y vale 5 */  
}
```

Dirección Contenido Gráfica

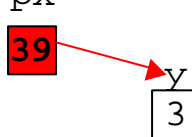
px-> 35:

?	?	?	?
0	0	0	3



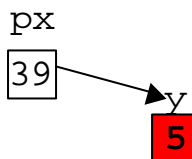
px-> 35:

0	0	0	39
0	0	0	3



px-> 35:

0	0	0	39
0	0	0	5



Punteros

La declaración de punteros genéricos a direcciones se asocian al tipo `void`.

Declarar una variable (que no sea un puntero) de tipo `void` no tiene sentido.

Ejemplo:

```
void *px, v;    /* La variable v
                  está mal
                  declarada */
```

Casting

- *Casting*: mecanismo usado para cambiar de tipo expresiones y variables:

```
int a;
```

```
float b;
```

```
char c;
```

```
b=65.0;
```

```
a=(int)b;    /* a vale 65 */
```

```
c=(char)a;   /* c vale 65 (Código ASCII  
de 'A') */
```

Programación en C

Operaciones aritméticas

Operaciones aritméticas

- El operador de asignación es el igual (=).
- Los operadores aritméticos son:
 - Suma (+)
 - Resta (-)
 - Multiplicación (*)
 - División (/)
 - Módulo o resto (%)

Operaciones aritméticas

- División entera vs división real:

- Depende de los operandos:

4 / 3 --> 1 entero

4.0 / 3 --> 1.333 real

4 / 3.0 --> 1.333 real

4.0 / 3.0 --> 1.333 real

Pre/post-incrementos

Los operadores unarios (++) y (--) representan operaciones de incremento y decremento, respectivamente.

`a++; /* similar a a=a+1 */`

Ejemplos:

`a=3; b=a++; /* a=4, b=3 */`

`a=3; b=++a; /* a=4, b=4 */`

`a=3; b=a--; /* a=2, b=3 */`

Operaciones de asignación

El operador de asignación en C es el igual(=)

```
a=b+3 ;
```

Existen otras variantes de asignación:

```
a+=3;    /* Equivalente a a=a+3 */
```

```
a*=c+d;  /* Equivalente a a=a*(c+d) */
```

```
a/=a+1;  /* Equivalente a a=a/(a+1) */
```

Para las asignaciones entre variables o expresiones de tipos diferentes se recomienda hacer *casting*:

```
a=(int)(x/2.34);
```

Programación en C

Sentencias de control

Operadores de comparación

Los operadores de comparación en C son:

- Igual (==)
- Distinto (!=)
- Mayor (>) y Mayor o igual (>=)
- Menor (<) y Menor o igual (<=)

El resultado de un operador de comparación es un valor entero (0 es falso) y (distinto de 0 verdadero).

```
a=3>7 /* a vale 0 (falso) */
```

Operadores lógicos

Sobre expresiones booleanas (enteros) se definen los siguientes operadores lógicos:

- And lógico (&&)
- Or lógico (||)
- Negación lógica (!)

Ejemplo

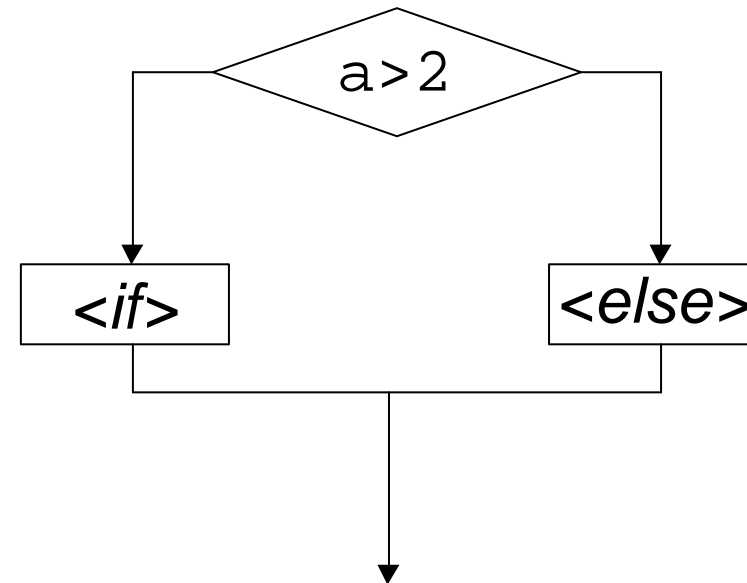
```
a = ( 3 > 2 || 5 == 4 ) && !1 /* Falso */
```

C tiene un modelo de evaluación perezoso.

```
a = 3 > 2 || w == 4 /* w == 4 no se evalúa */
```

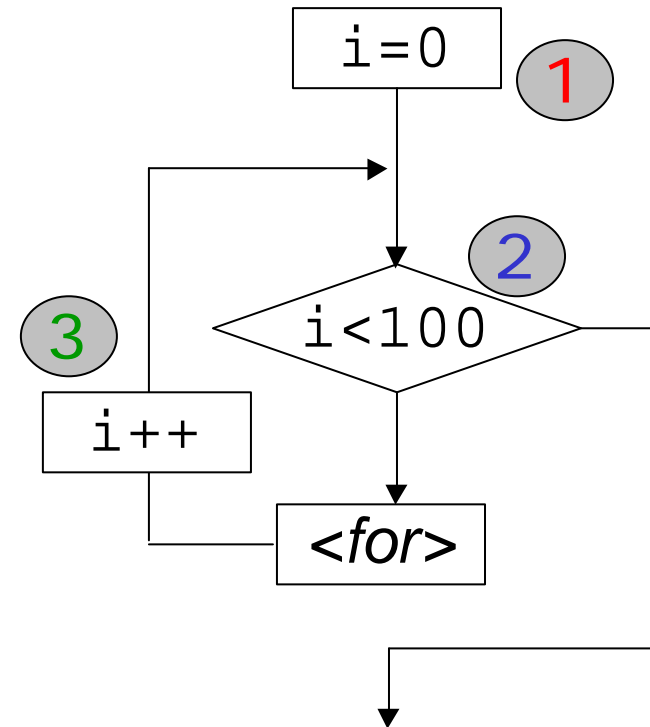
if ... else

```
int main()  
{  
    int a=3,b;  
  
    if(a>2)  
    {  
        b=100+a;  
        printf("parte if");  
    }  
    else  
        printf("parte else");  
}
```



for

```
int main()  
{  
    int i,ac=0;  
  
    for(i=0;i<100;i++)  
    {  
        printf("%d",i*i);  
        ac+=i;  
    }  
}
```

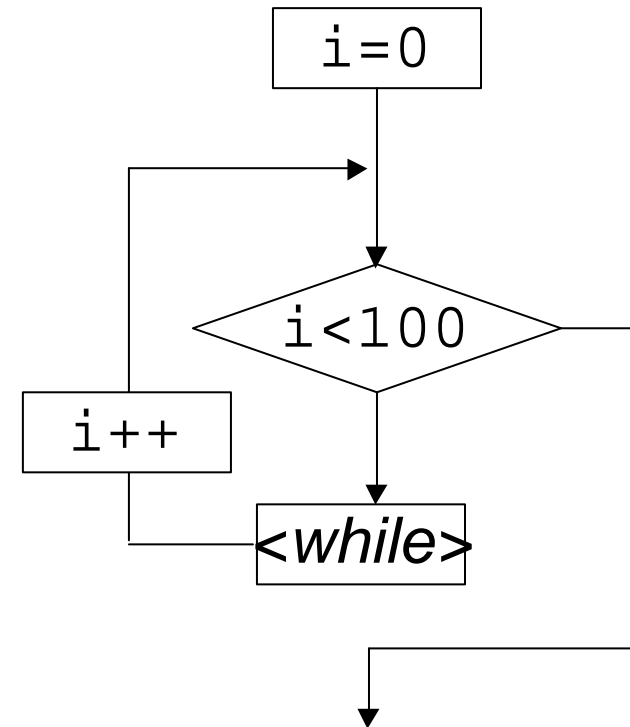


Sintaxis:

for(**inicialización**, **condición_permanencia**, **incremento**)

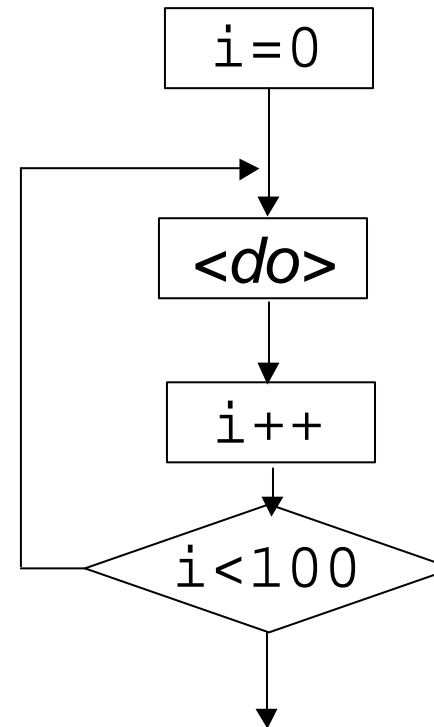
while

```
int main()  
{  
    int i=0,ac=0;  
  
    while(i<100)  
    {  
        printf("%d",i*i);  
        ac+=i;  
        i++;  
    }  
}
```



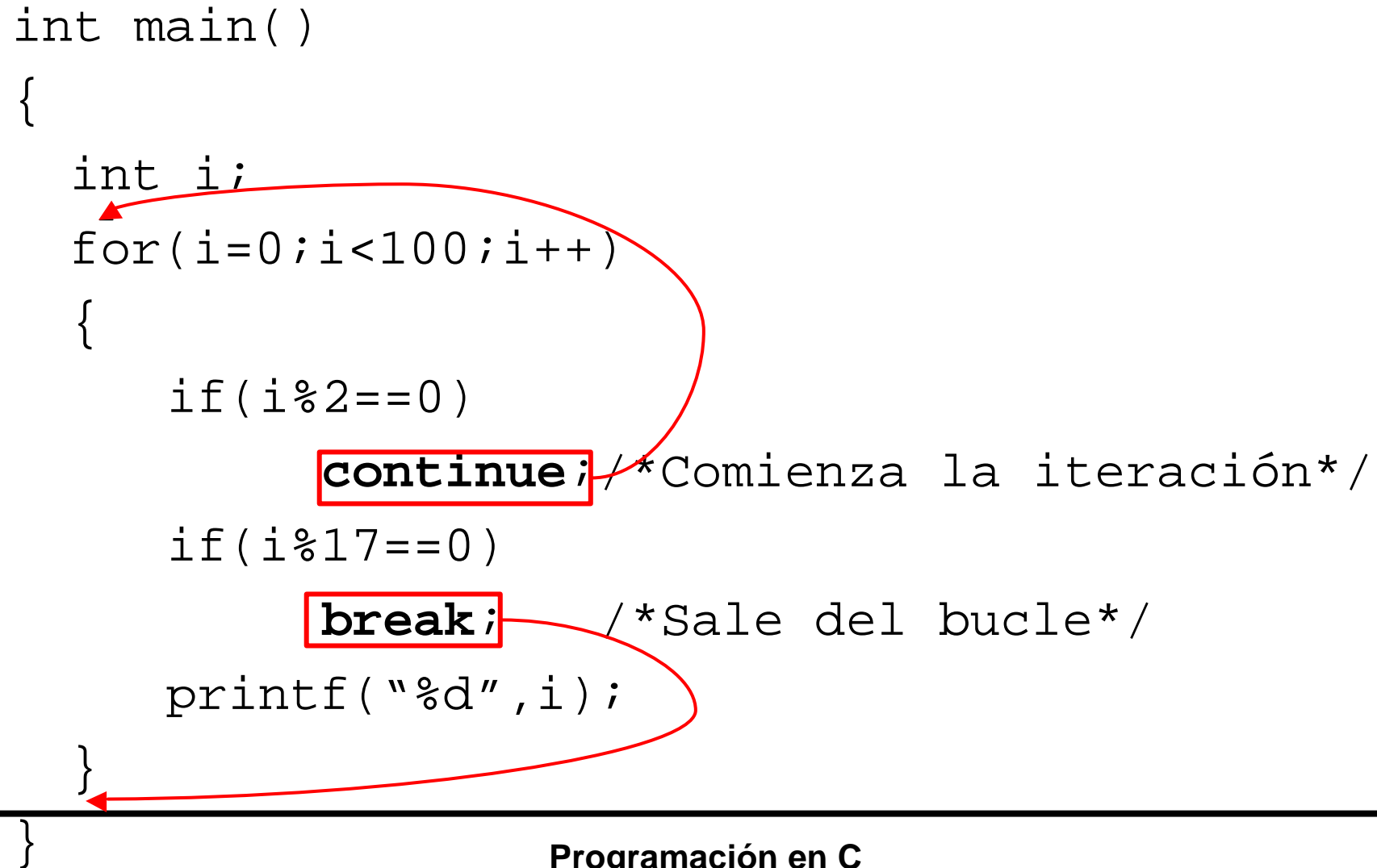
do ... while

```
int main()  
{  
    int i=0,ac=0;  
  
    do  
    {  
        printf( "%d",i*i);  
        ac+=i;  
        i++;  
    }  
    while(i<100);  
}
```



break y continue

```
int main( )
{
    int i;
    for(i=0;i<100;i++)
    {
        if(i%2==0)
            continue; /*Comienza la iteración*/
        if(i%17==0)
            break; /*Sale del bucle*/
        printf("%d",i);
    }
}
```



switch

```
switch( ch )
{
    case 'A': printf( "A" );
               break;

    case 'B':
    case 'C': printf( "B o C" );
    case 'D': printf( "B, C o D" );
               break;

    default:   printf( "Otra letra" );
}
}
```

Operador ?

```
int main( )
{
    int a,b=4,c=5;

    a=b>0 ? c : c+1;
    /* Equivalente a
        if(b>0)
            a=c;
        else
            a=c+1;    */
}
```

Programación en C

Arrays y Strings

Definición de Arrays

La definición de una variable de tipo array (vector) se realiza indicando la dimensión entre corchetes:

```
int a[100]; /* Un vector de 100 enteros */  
float vx[4][4]; /* Matriz de 4x4 reales */  
int *pt[10][10][10][10]; /* Una matriz de  
4 dimensiones de punteros a enteros */
```

Asimismo, pueden inicializarse:

```
float a[3]={2.45, -1.34, 2.11};  
int vx[2][3]={ {3,5,1},  
               {2,1,2} };
```


Indexación de arrays

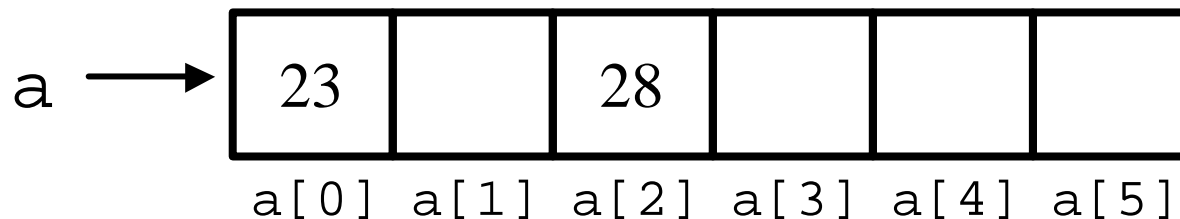
A diferencia de otros lenguajes los arrays en C comienzan por el elemento 0 y terminan en el n-1.

```
int a[6];
```

```
a[0]=23;
```

```
a[3]=a[0]+5;
```

```
for(i=0;i<6;i++) printf("%d",a[i]);
```



Strings

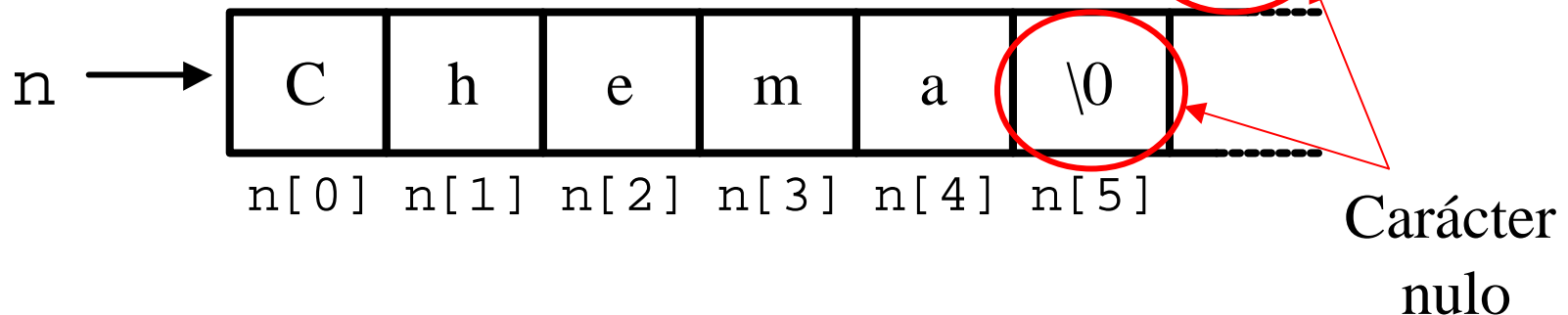
Los strings son los arrays de caracteres de una dimensión. Son las cadenas de caracteres.

Definición:

```
char x[20], n[50] = "Chema";
```

*/*equivalente a*

```
char n[50] = { 'C', 'h', 'e', 'm', 'a', '\0' } */
```



Asignación de Strings

La asignación de strings por medio del operador (=) sólo es posible en la declaración.

Ejemplo:

```
char n[50]="Chema";    /* Correcto */  
n="Otro nombre";      /* Error: no es  
                        declaración */
```

Para las operaciones sobre strings se utilizan diferentes funciones de librería. Por ejemplo, **strlen()** calcula el tamaño del string (número de caracteres).

Arrays y punteros

El identificador de una variable array tiene el valor de la dirección de comienzo del mismo. Por lo tanto, su valor puede usarse como un puntero.

```
int *pb, *pc;
```

```
int a[5]={10,20,30,40,50};
```

```
pb=a;
```

```
*pb=11;
```

```
pc=&a[3];
```

```
*pc=44;
```

a [10|20|30|40|50]

pb pc

a [11|20|30|40|50]

pb pc

a [11|20|30|44|50]

pb pc

Arrays y punteros

Los arrays de varias dimensiones sí se diferencian de los punteros múltiples:

Matriz de 2 dimensiones:

```
int mx[3][3];
```

mx

0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2
-----	-----	-----	-----	-----	-----	-----	-----	-----

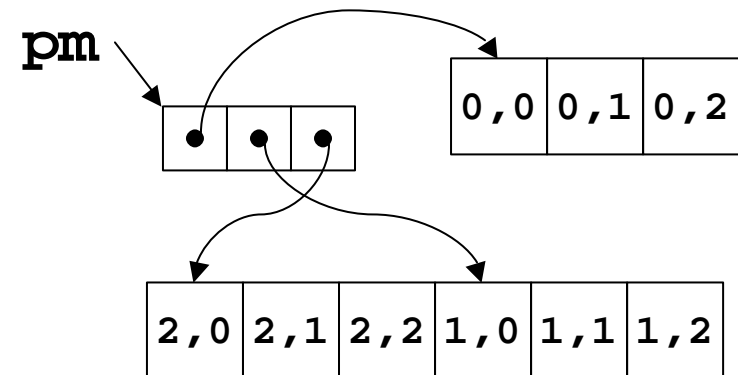
```
pm=mx; /* ERROR */
```

```
pm[0]=mx[1]; /* OK */
```

```
Pm[0][1]=mx[1][2] /* OK */
```

Puntero múltiple:

```
int **pm;
```



Programación en C

Funciones

Definición de una función

La definición de una función tiene la estructura:

```
tipo identificador (argumentos ...)  
{  
    ...  
    cuerpo de la función  
    ...  
}
```

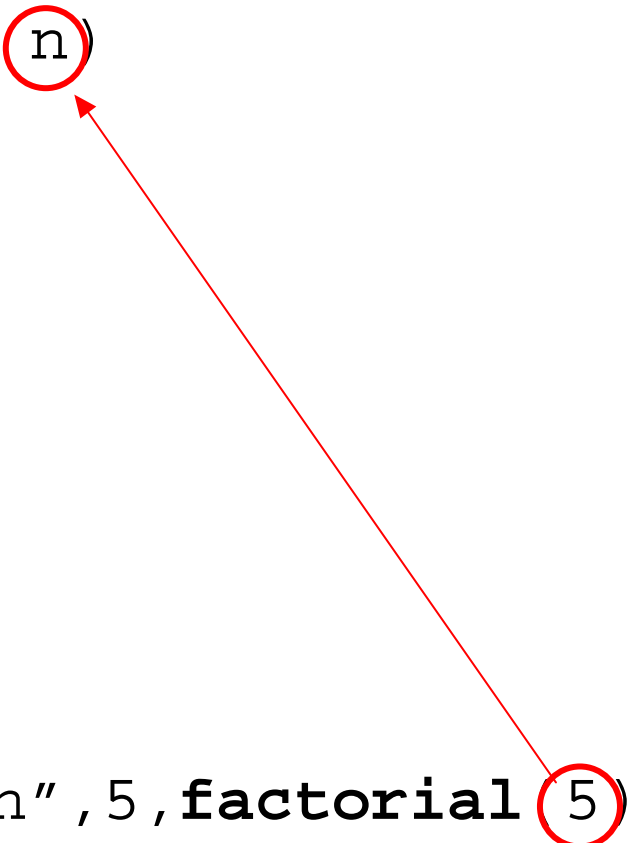
Uso de una función

Una función se invoca proporcionando valores a los argumentos de la llamada.

- Los argumentos se pasan **siempre por valor**.
- El valor se devuelve por medio de `return ()`.
- Los procedimientos son funciones de tipo `void`.
- El control del número y tipo de argumentos es mínimo.
- Las funciones en C admiten recursividad.

Función de ejemplo

```
int factorial(int n)
{
    int ret=1;
    while (n>1)
        ret*=n--;
    return(ret);
}
int main()
{
    printf( "%d!=%d\n" , 5 , factorial 5 ) ;
}
```



Declaración de funciones

Para poder hacer uso de una función es necesario que ésta esté definida o declarada con antelación.

- Definición de la función: Todo el código de la función.
- Declaración de la función: Únicamente la cabecera o prototipo de la función:

```
int factorial(int n);
```

```
int factorial(int);
```

```
int factorial();
```

Prototipo de una función

```
int factorial(int n);      /* Prototipo */
```

```
int main()  
{  
    printf( "%d!=%d\n" , 5 , factorial( 5 ) );  
}
```

```
int factorial(int n)      /* Definición */  
{  
    ...  
}
```

Paso por referencia

El lenguaje C **NO** tiene paso por referencia. En su lugar se pasa por valor la dirección de la variable a modificar.

```
int reiniciar(int *a, int b)
{
    *a=0; b=0;
}
```

```
int x=2,y=2;
reiniciar(&x,y); /* x valdrá 0 e y 2 */
```

Paso por referencia

```
int x=2,y=2;
```

```
reiniciar(&x,y);
```

x **2** y **2**

x **2** y **2**
a **•** b **2**

```
int reiniciar(int *a, int b)
```

```
{
```

```
    *a=0; b=0;
```

```
}
```

x **0** y **2**
a **•** b **0**

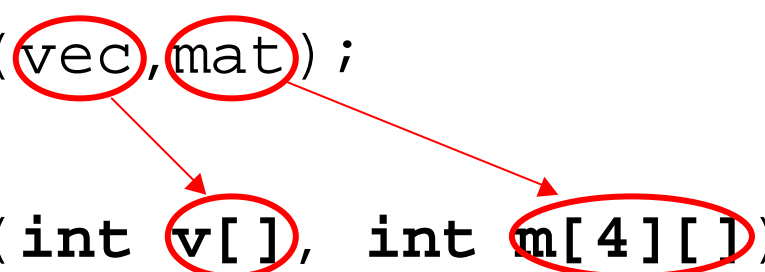
Argumentos de tipo array

Cuando un array se pasa como argumento a una función, la última de las dimensiones no se define.

Ejemplo:

```
int vec[12], mat[4][4];  
calcula(vec, mat);
```

```
void calcula(int v[], int m[4][ ])
{
    ...
}
```



Programación en C

Estructuras de datos

Tipos de estructuras

El lenguaje C tiene tres tipos de estructuras de datos:

- Registro o estructura (`struct`).
- Unión de campos (`union`).
- Tipo enumerado (`enum`).

struct

Un `struct` es un tipo de datos complejo conformado por un conjunto de campos de otros tipos (básicos o complejos) asociados a un identificador:

```
struct [etiqueta]  
{  
    tipo campo;  
    tipo campo;  
    ...  
};
```

struct

```
struct persona
{
    char  nombre[20];
    int   edad;
    float peso;
} yo,tu,ellos[10];
```

```
struct persona el={"Antonio López",31,80};
struct persona *ella, todos[20];
```

struct

El acceso a los campos de una estructura se hace por medio de un punto (.) o de una flecha (->) si es un puntero a estructura.

```
struct persona el, *ella, todos[20];
```

```
printf("Su nombre %s\n", el.nombre);
```

```
todos[2].edad=20;
```

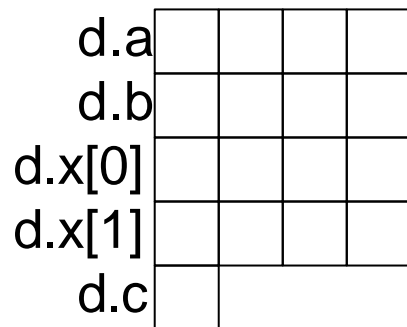
```
ella=&todos[2];
```

```
printf("La edad de ella es %d\n", ella->edad);
```

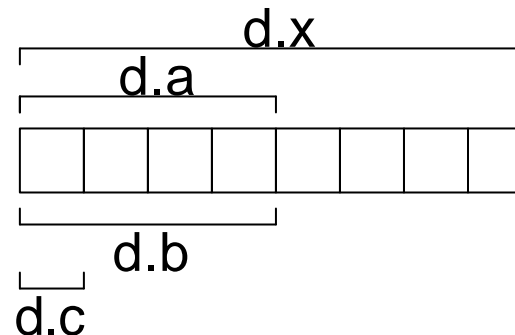
union

Un `union` es similar a un `struct`, pero todos los campos comparten la misma memoria.

```
struct datos
{
    int a,b;
    int x[2];
    char c;
} d;
```



```
union datos
{
    int a,b;
    int x[2];
    char c;
} d;
```



Uso de union

Los union se usan para diferentes representaciones de los datos o para información condicionada:

union

```
{  
    int    integer;  
    char oct[4];  
} data;
```

```
struct empleado  
{  
    char nombre[40];  
    int  tipo_contrato;  
    union  
    {  
        int    nomina;  
        int    pts_hora;  
    } sueldo;  
} p;
```

Estructuras y funciones

Las estructuras de datos son tipos complejos y (aunque ciertos compiladores lo admiten) no deben ser pasados como argumentos ni devueltos por funciones. En su lugar se usan punteros a dichas estructuras:

```
void evaluar_empleado(struct empleado* emp);  
struct empleado* nuevo_empleado();
```

enum

Las enumeraciones con conjuntos de constantes numéricas definidas por el usuario.

```
enum color {rojo, verde, azul} fondo;
```

```
enum color letras, borde=verde;
```

```
enum tipo_empleado {contratado=1,  
                    temporal=2,  
                    becario=3};
```

Definición de nuevos tipos

Las sentencias `typedef` se usan para definir nuevos tipos en base a tipos ya definidos:

```
typedef int boolean;  
typedef struct persona persona_t;  
typedef struct punto  
{  
    int          coord[3];  
    enum color col;  
} punto_t;  
persona_t p[4];
```

Programación en C

Entrada/salida básica

Funciones de entrada/salida

- Las funcionalidades de entrada/salida en C no pertenecen a las palabras reservadas del lenguaje. Son funciones de librería, por ejemplo:
 - Entrada: `scanf ()`.
 - Salida: `printf ()`.

printf ()

- El formato de llamada de `printf ()` es:

`printf(format, exp1, exp2, exp3, ..., expn);`

donde:

- *format* : Es el string de formato de salida de los datos.
- *exp_i* : Es la expresión a incluir dentro del formato.

printf()

Ejemplo:

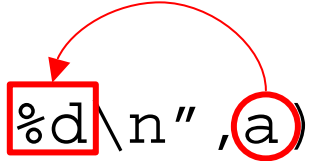
```
int a=3;
```

```
float x=23.0;
```

```
char c='A';
```

```
printf("Hola mundo!!\n");
```

```
printf("Un entero %d\n", a);
```



A red box highlights the format specifier `%d` and a red circle highlights the argument `a`. A red curved arrow points from the circle around `a` to the box around `%d`.

```
printf("Un real %f \ny un char %c\n", x, c);
```



Red boxes highlight the format specifiers `%f` and `%c`. Red circles highlight the arguments `x` and `c`. A red curved arrow points from the circle around `x` to the box around `%f`. Another red curved arrow points from the circle around `c` to the box around `%c`.

printf ()

Formato	Expresión	Resultado
%d %i	entero	entero decimal con signo
%u	entero	entero decimal sin signo
%o	entero	entero octal sin signo
%x %X	entero	entero hexadecimal sin signo
%f	real	real en notación punto
%e %E %g %G	real	real en notación científica
%c	carácter	carácter
%p	puntero	dirección de memoria
%s	string	cadena de caracteres
%ld %lu ...	entero largo	entero largo (distintos formatos)

`printf ()`

- Otras opciones de formato:
 - Precisión (número de decimales).
 - Justificación (izquierda o derecha).
 - Caracteres especiales (% o \).
 - ...

Ver página del manual:

`man printf`

scanf ()

- El formato de llamada de `scanf ()` es:

`scanf(format, dir1, dir2, dir3, ..., dirn);`

donde:

- *format* : Es el string de formato de entrada de los datos.
- *dir_i* : Es la dirección donde se almacena el resultado.

scanf ()

Ejemplo

```
int a,*pa;
```

```
float x;
```

```
char c;
```

```
scanf("%d",&a); /* Lee un entero y lo  
                  almacena en a */
```

```
scanf("%f %c",&x,&c); /* Lee x y c */
```

```
scanf("%d",pa); /* PELIGROSO */
```

```
pa=&a; scanf("%d",pa); /* OK. Lee a */
```


scanf () Lectura de strings

Ejemplo:

```
char *pc;
```

```
char str[82];
```

```
scanf( "%s" ,pc ); /* PELIGROSO */
```

```
scanf( "%s" ,str ); /* Lee hasta un blanco o  
fin de línea */
```

```
scanf( "%[^\n]" ,str ); /* Lee toda la línea */
```

Programación en C

Ejemplos I

Ejemplos I-1

Se plantea el desarrollo de un programa que lea los datos (nombre, apellidos, nif y sueldo) de 10 empleados por teclado e imprima los empleados con sueldo máximo y mínimo, así como la media de los sueldos.

Ejemplos I-1: Solución

En primer lugar definimos la estructura de datos a utilizar.

```
struct empleado_st
{
    char nombre[40];
    char apellidos[40];
    char nif[10];
    long sueldo;
};
```

```
typedef struct empleado_st empleado_t;
```

Ejemplos I-1: Solución

Seguidamente programamos el cuerpo del programa principal.

```
int main()  
{  
    empleado_t emp[10];  
    int i,min,max;  
    float med;  
  
    for(i=0;i<10;i++)  
        leer_empleado(&emp[i]);  
  
    min=buscar_min(emp,10);  
    max=buscar_max(emp,10);  
    med=buscar_med(emp,10);  
  
    printf("Mínimo:\n");  
    imprimir_empleado(&emp[min]);  
    printf("Máximo:\n");  
    imprimir_empleado(&emp[max]);  
    printf("Media:%9.2f\n",  
           med);  
    return(0);  
}
```

Ejemplos I-1: Solución

Los prototipos de las funciones a implementar son:

```
void  leer_empleado      (empleado_t *pe) ;  
int   buscar_min        (empleado_t es[],  
                          int        tam) ;  
int   buscar_max        (empleado_t es[],  
                          int        tam) ;  
float buscar_med        (empleado_t es[],  
                          int        tam) ;  
void  imprimir_empleado (empleado_t *pe) ;
```

Ejemplos I-1: Solución

```
void leer_empleado (empleado_t *pe)
{
    printf("Nombre y Apellidos: ");
    scanf("%s %[^\n]",
          pe->nombre, pe->apellidos);
    printf("NIF: ");
    scanf("%s", pe->nif);
    printf("Sueldo: ");
    scanf("%ld", &pe->suelo);
    printf("-----\n");
}
```

Ejemplos I-1: Solución

```
int    buscar_min          (empleado_t es[],
                              int        tam)
{
    int candidato=0,i;

    for(i=1;i<tam;i++)
        if(es[i].sueldo<es[candidato].sueldo)
            candidato=i;
    return(candidato);
}
```


Ejemplos I-1: Solución

```
int    buscar_max          (empleado_t es[],
                              int        tam)
{
    int candidato=0,i;

    for(i=1;i<tam;i++)
        if(es[i].sueldo>es[candidato].sueldo)
            candidato=i;
    return(candidato);
}
```

Ejemplos I-1: Solución

```
float buscar_med      (empleado_t es[],
                       int      tam)
{
    int i;
    float acc=0;
    for(i=0;i<tam;i++)
        acc+=(float)es[i].sueldo;
    return(acc/(float)tam);
}
```

Ejemplos I-1: Solución

```
void  imprimir_empleado(empleado_t *pe)
{
    printf( "\tNombre:      %s\n", pe->nombre );
    printf( "\tApellidos:  %s\n", pe->apellidos );
    printf( "\tNIF:        %s\n", pe->nif );
    printf( "\tSueldo:       %ld\n", pe->sueldo );
}
```